Introduction to Groovy

Paul Woods

mr.paul.woods@gmail.com

@mr_paul_woods

https://github.com/paulwoods

3/5/2014

https://github.com/paulwoods/intro-to-groovy-2014

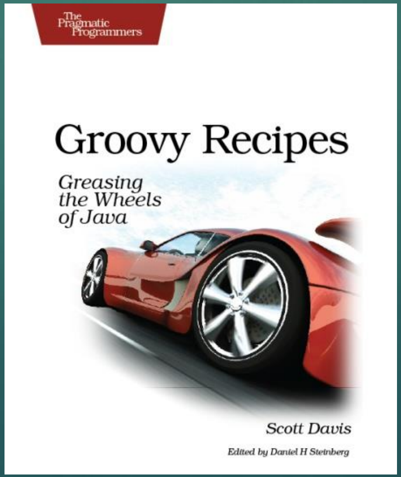
Table Of Contents

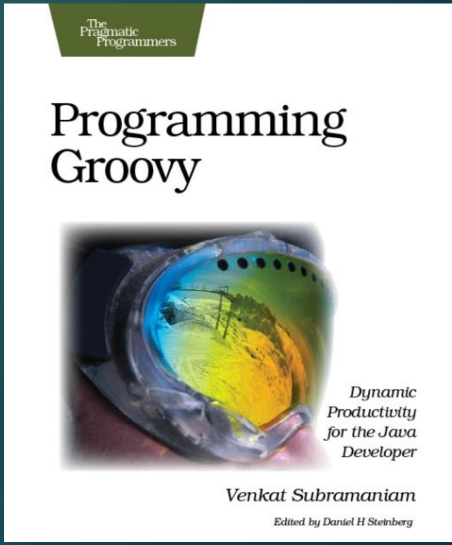
Teaching Resources

- Mr. Haki - Great Groovy Tips Resource
  - http://mrhaki.blogspot.com/
- How to do X in Groovy
  - http://groovy-almanac.org/
- Share and Run Groovy Scripts
  - http://groovyconsole.appspot.com/
- Learn Groovy by Fixing Failing Tests
  - http://groovykoans.org/

Books

- Groovy In Action
- Programming Groovy
- Groovy Recipes


The Pragmatic Programmers
Groovy Recipes
Greasing the Wheels of Java
Scott Davis
Edited by Daniel H Steinberg


The Pragmatic Programmers
Programming Groovy
Dynamic Productivity for the Java Developer
Venkat Subramaniam
Edited by Daniel H Steinberg

Groovy - Installing to Your Computer

- Windows Users
  - Install Java JDK
  - Set JAVA_HOME
  - Make sure <javahome>/bin is in path
  - Download Zip or installer package
    - http://groovy.codehaus.org/Download
  - Make sure <groovyhome>/bin is in path
  - Test it
    - java -version
    - javac -version
  - groovy –version

- Linux users should check out the GVM tool
  - http://gvmtool.net/

Groovy - Running on Web Console

- http://groovyconsole.appspot.com/

- Allows you to execute groovy programs on a web-based console.

Hello World – Java

- Create a file called Hello.java

  ```
  public class Hello {
     public static void main(String[] args) {
             System.out.println("Hello Java World");
     }
  }
  ```

- Compile It
  - javac Hello.java
- Run It
  - java Hello

http://groovyconsole.appspot.com/script/2415001

Hello World - Groovy

- Create a file called hello.groovy
  - println "Hello Groovy World!"
- Execute the file
  - groovy hello.groovy

Improvements over Java

- Reduced ceremony
  - No Class
  - No Public classifier
  - No System.out.println
  - No Parenthesis
  - No Semicolon
  - No Main method
- * Of course, we can still add them if we want, and in some cases they are required.

- Default Imports
  - java.io.*
  - java.lang.*
  - java.math.BigDecimal
  - java.math.BigInteger
  - java.net.*
  - java.util.*
  - groovy.lang.*
  - groovy.util.*

Basics - Primitives

- There are no primitive types in Groovy. They are promoted to objects.
  - int a = 1
  - float f = 2.5
  - println a.class
  - println f.class
    - class java.lang.Integer
    - class java.lang.Float

http://groovyconsole.appspot.com/script/2425001

Basics - Types

- You are not forced to define the types, either. The def keyword allows you to do this:
  - def a = 1
  - def b = 2.0
  - def c = "Howdy"
  - println a.class
  - println b.class
  - println c.class
    - class java.lang.Integer
    - class java.math.BigDecimal
    - class java.lang.String

http://groovyconsole.appspot.com/script/2435001

Basics - String Quotes

- Strings can use single quotes or double quotes.
  - def a = "Hello World"
  - def b = 'Goodbye'
  - println a
  - println b
  - println a.class
  - println b.class
    - Hello World
    - Goodbye
    - class java.lang.String
    - class java.lang.String

http://groovyconsole.appspot.com/script/2445001

Basics - GStrings

- The GString type allows expressions to be evaluated inside a string. Simple variables/properties are prefixed with $. Full expressions are wrapped in ${}
  - def first = "Paul"
  - def last = "Woods“
  - def a = 11
  - def b = 22
  - def fullname = "$first $last ${a + b}"
  - println fullname
  - println “escape \$”
    - Paul Woods 33
    - escape $

http://groovyconsole.appspot.com/script/2455001

Basics - GStrings

- GStrings are not Strings.
- They are very similar and can be used almost anywhere a String can be used.
  - Don’t use them as map keys.

Basic - GStrings - Double Quotes

- GString uses double quote marks in order to evaluate expressions. Single quote marks are not evaluated.
  - def age = 8
  - def a = "I am $age years old"
  - def b = 'I am $age years old'
  - println "a = " + a
  - println "b = " + b
    - a = I am 8 years old
    - b = I am $age years old

http://groovyconsole.appspot.com/script/2465001

Basics - String - Triple Quotes

- You can use triple quotes to keep from escaping quote marks.
  - def a = """She said "Bring home some milk", but I forgot"""
  - def b = '''It's time to learn Groovy.'''
  - println a
  - println b
    - She said "Bring home some milk", but I forgot.
    - It's time to learn Groovy.

http://groovyconsole.appspot.com/edit/2465002

Basics - Multi-line Strings

- You can easily define multi-line strings. Each line of the string is terminated with a newline - \n
  - def data = """
  - This
  - Is
  - A
  - Test"""
  - println data
    - This
    - Is
    - A
    - Test

http://groovyconsole.appspot.com/script/2425002

Basics - String Comparison

- Groovy overrides the == for case-sensitive String comparison.
- No more: if(0 == a.compareTo(b)) …
  - def a = "abc"
  - def b = "a" + "b" + "c"
  - println a == b
    - true

http://groovyconsole.appspot.com/script/2425003

Basics - Script Files

- Groovy files can be either compiled into classes, or ran as script. Here is a script that defines a function, then executes it.
  - Integer sum(Integer a, Integer b) {
  -   a + b
  - }
  - println "10 + 20 is " + sum(10,20)
    - 10 + 20 is 30

http://groovyconsole.appspot.com/script/2485001

Basics - Groovy Truth

- Groovy truth tells how values are coerced into Booleans.
  - non zero numbers are true.
  - zero numbers are false
  - non-null objects are true
  - null objects are false
  - non-empty strings are true
  - empty strings are false
  - non-empty collections are true
  - empty collections are false

Basics - Class - Map Constructor

- A map default constructor is available to classes.
  - class Name {
  -   String first
  -   String last
  - }
  - def name = new Name(first:"Paul", last:"Woods")
  - println name.dump()
    - <Name@5c6ed020 first=Paul last=Woods>

http://groovyconsole.appspot.com/script/2435002

Basics - Class - Getters & Setters

- For all public fields, groovy makes the field private, and automatically creates a getter and setter for each.
- If you define a getter and/or setter, groovy doesn't define one.
- If you mark the field as readonly, only a getter will be created

Lists

- Groovy has a native syntax for lists. It uses square brackets [] to indicate a list.
- Define a empty list
  - def list = []
- Initialize a list with values
  - def list = [1, 2, 3]
  - println list
    - [1, 2, 3]

http://groovyconsole.appspot.com/script/2425004

Lists

- Lists are a normal java data type
  - def list = []
  - println list.class
    - class java.util.ArrayList

List Gotcha - Static Type Checking

- The items in a list are non-typed.
- We add a Integer, BigDecimal, String and Date
  - def a = [1, 2.3, "Abc", new Date()]
  - println a
    - [1, 2.3, Abc, Tue Sep 25 20:21:17 CDT 2012]
- Generics are compile-time checked. Groovy ignores them, with no syntax error
  - List<Integer> b = [“A”, new Date()]
  - println b
    - [A, Tue Sep 25 20:22:10 CDT 2012]

List - Adding Elements

- Add elements to a list using left shift
  - def list = ['q', 'w', 'e']
  - list << 'r' << 't' << 'y'
  - println list
    - [q, w, e, r, t, y]
- Add a list to a list - use the plus operation
  - def list = ["a","b","c"]
  - list += [1,2,3]
  - println list
    - [a, b, c, 1, 2, 3]

http://groovyconsole.appspot.com/script/2445002

List - Iterating 1

- Multiple ways to loop through the elements in a list.
- By For Colon
  - def list = ['q', 'w', 'e', 'r', 't', 'y']
  - for(def item : list) {
  -    println "by item : $item"
  - }
    - by item : q
    - by item : w
    - by item : e
    - by item : r
    - by item : t
    - by item : y

http://groovyconsole.appspot.com/script/2405002

List - Iterating 2

- By For in
  - def list = ['q', 'w', 'e', 'r', 't', 'y']
  - for(def item in list) {
  -   println "by item in : $item"
  - }
    - by item in : q
    - by item in : w
    - by item in : e
    - by item in : r
    - by item in : t
    - by item in : y

http://groovyconsole.appspot.com/script/2405003

List - Iterating 3

- Use the each method on list
  - def list = ['q', 'w', 'e', 'r', 't', 'y']
  - list.each { item ->
  -     println "by each : $item"
  - }
    - by each : q
    - by each : w
    - by each : e
    - by each : r
    - by each : t
    - by each : y

http://groovyconsole.appspot.com/script/2495001

List - Transform

- Transform a list into another list using the collect method. The results of the closure are appended in a new list.
  - def list1 = [1,2,3,4,5]
  - def list2 = list1.collect { it * 10 }
  - println "list1=$list1"
  - println "list2=$list2"
    - list1=[1, 2, 3, 4, 5]
    - list2=[10, 20, 30, 40, 50]

http://groovyconsole.appspot.com/script/2405004

List - Retrieving Elements

- Multiple ways to retrieve list elements
- def list = ['q', 'w', 'e', 'r', 't', 'y']
  - println "element 0 : ${list.get(0)}"
  - println "element 1 : ${list[1]}"
  - println "elements 1,3,5 : ${list[1,3,5]}"
  - println "elements 0..3 : ${list[0..3]}"
  - println "last 3 elements : ${list[-3..-1]} "
    - element 0 : q
    - element 1 : w
    - elements 1,3,5 : [w, r, y]
    - elements 0..3 : [q, w, e, r]
    - last 3 elements : [r, t, y]

http://groovyconsole.appspot.com/script/2505001

List - Removing Elements

- Removing Elements
  - def list = ["q", "w", "e", "r", "t", "y"]
  - println list
  - list.remove(0)
  - println list
  - list -= "e"
  - println list
  - list -= ["t", "r"]
  - println list
    - [q, w, e, r, t, y]
    - [w, e, r, t, y]
    - [w, r, t, y]
    - [w, y]

http://groovyconsole.appspot.com/script/2515001

List - Sorting 1

- Sorting Lists - By Default, the original list is modified.
  - def original = ['q', 'w', 'e', 'r', 't', 'y']
  - def sorted = original.sort()
  - println "original = " + original
  - println "sorted   = " + sorted
    - original = [e, q, r, t, w, y]
    - sorted   = [e, q, r, t, w, y]

http://groovyconsole.appspot.com/script/2415002

List - Sorting 2

- Sorting Lists - sort(false) will not sort the original.
  - def original = ['q', 'w', 'e', 'r', 't', 'y']
  - def sorted = original.sort(false)
  - println "original = " + original
  - println "sorted   = " + sorted
    - original = [q, w, e, r, t, y]
    - sorted   = [e, q, r, t, w, y]

http://groovyconsole.appspot.com/script/2455002

List - Unique 1

- Retrieving the unique elements.
- The list does not need to be sorted.
- The original list is modified.
  - def original = ['a', 'b', 'c', 'a', 'b', 'c']
  - def uniqued = original.unique()
  - println "original = " + original
  - println "uniqued = " + uniqued
    - original = [a, b, c]
    - uniqued = [a, b, c]

http://groovyconsole.appspot.com/script/2475002

List - Unique 2

- Use .unique(false) to not modify the original.
  - def original = ['a', 'b', 'c', 'a', 'b', 'c']
  - def uniqued = original.unique(false)
  - println "original = " + original
  - println "uniqued = " + uniqued
    - original = [a, b, c, a, b, c]
    - uniqued = [a, b, c]

http://groovyconsole.appspot.com/script/2425005

List - Find

- Finding a single element in a list
  - def list = ['q', 'w', 'e', 'r', 't', 'y']
  - def item1 = list.find { item -> item == 'w' }
  - def item2 = list.find { item -> item == '12345' }
  - println "find 1 : " + item1
  - println "find 2 : " + item2
    - find 1 : w
    - find 2 : null

http://groovyconsole.appspot.com/script/2435003

List - FindAll

- Finding all matching elements in in a list
  - def list = ['q', 'w', 'e', 'r', 't', 'y']
  - def letters = list.findAll { it < 't' }
  - println "findAll : $letters"
    - findAll : [q, e, r]

http://groovyconsole.appspot.com/script/2445003

List - Every

- Returns true if the closure returns true for every item in the list.
  - def list = [4,5,6]
  - boolean result1 = list.every { item -> item < 10 }
  - println "every item less than 10 : $result1"
  - boolean result2 = list.every { item -> 0 == item%2 }
  - println "every item is even : $result2"
    - every item less than 10 : true
    - every item is even : false

http://groovyconsole.appspot.com/script/2495002

List - Any

- Returns true if the closure returns true for any item in the list.
  - def list = [4,5,6]
  - boolean result1 = list.any { item -> item > 5 }
  - println "contains atleast one item greater than 5 : $result1"
  - boolean result2 = list.any { item -> 1 == item%2 }
  - println "contains atleast one item that is odd : $result2"
    - contains atleast one item greater than 5 : true
    - contains atleast one item that is odd : true

http://groovyconsole.appspot.com/script/2425006

List - Join

- convert list into a string by converting each element to a string (toString()) and inserting a delimiter between elements.
  - def list = ['q','w','e','r','t','y']
  - def result = list.join("-")
  - println result
  - println result.class
    - q-w-e-r-t-y
    - class java.lang.String

http://groovyconsole.appspot.com/script/2485002

List - Advanced 1

- Sorting by values in a Map
  - list = [
  -    [first:"paul", last:"woods"],
  -    [first:"linda", last:"zinde"],
  -    [first:"alex", last:"zinde"],
  -    [first:"paul", last:"allen"]
  -]
  - println "sorted by first : ${list.sort { it.first } }"
  - println "sorted by last : ${list.sort { it.last } }"
    - sorted by first : [[first:alex, last:zinde], [first:linda, last:zinde], [first:paul, last:woods], [first:paul, last:allen]]
    - sorted by last : [[first:paul, last:allen], [first:paul, last:woods], [first:alex, last:zinde], [first:linda, last:zinde]]

http://groovyconsole.appspot.com/script/2405005

List - Advanced 2

```
list = [
  [first:"paul", last:"woods"],
  [first:"linda", last:"zinde"],
  [first:"alex", last:"zinde"],
  [first:"paul", last:"allen"]
]
// sorting by a value in a map
println "sorted by first : ${list.sort { it.first } }"
println "sorted by last  : ${list.sort { it.last } }"
// sorting by 2 values
def sorted = list.sort { x, y ->
    (x.last <=> y.last) ?: (x.first <=> y.first)
}
println "sort by last and first : ${sorted}"
```

http://groovyconsole.appspot.com/script/2435004

List - Advanced 3

- Transform a list of lists to a quoted-field csv string

```
def list = [
    [ "first", "last" ],
    [ "paul", "woods"],
    [ "linda", "zinde"],
    [ "alex", "zinde"],
    [ "paul", "allen"]
]
def csv = list.collect { row ->
    row.collect { item ->
        "\"$item\""
    }.join(',')
}.join('\n')
println csv
```

```
"first","last"
"paul","woods"
"linda","zinde"
"alex","zinde"
"paul","allen"
```

http://groovyconsole.appspot.com/script/2495003

List - Mystery

- Why does this work?
  - List<String> z = new ArrayList<String>()
  - z << "A"
  - z << 1
  - z << new Date()
  - println z

  - ? because generics in java are checked at compile time, and groovy doesn't check

http://groovyconsole.appspot.com/script/2435005

Map

- A map is defined using the [:] syntax
  - def name = [:]
- A map is a normal java data structure
  - def name = [:]
  - println name.getClass()
    - class java.util.LinkedHashMap

Map - Initialize with Data

- Create map
  - def map = [first : "Paul",last : "Woods"]
  - println map
    - [first:Paul, last:Woods]
- Tip - if you need iterate through your keys in order, do this:
  - def map = new TreeMap<String,String>()

Map - Add Data

- Add elements to a map
  - def map = [:]
  - map += [first : "Paul"]
  - map.middle = "Alexander"
  - map.put("last", "Woods")
  - println map
    - [first:Paul, middle:Alexander, last:Woods]

http://groovyconsole.appspot.com/script/2435006

Map - Iterating with For

- Looping through maps
  - def map = [first: "Paul", last: "Woods"]
  - for(keyValue in map) {
  -   println "keyValue = $keyValue"
  -   println "key   = $keyValue.key"
  -   println "value   = $keyValue.value"
  - }
    - keyValue = first=Paul
    - key   = first
    - value   = Paul
    - keyValue = last=Woods
    - key   = last
    - value   = Woods

http://groovyconsole.appspot.com/script/2475003

Map - Iterating with Each 1

- looping through maps
  - def map = [first: "Paul", last: "Woods"]
  - map.each { keyValue ->
  -    println "keyValue = $keyValue"
  -    println "key   = $keyValue.key"
  -    println "value   = $keyValue.value"
  - }
    - keyValue = first=Paul
    - key   = first
    - value   = Paul
    - keyValue = last=Woods
    - key   = last
    - value   = Woods

http://groovyconsole.appspot.com/script/2495004

Map - Iterating with Each 2

- Looping through maps. Closure has 2 parameters
  - def map = [first: "Paul", last: "Woods"]
  - map.each { key, value ->
  -   println "key   = $key"
  -   println "value = $value"
  - }
    - key   = first
    - value = Paul
    - key   = last
    - value = Woods

http://groovyconsole.appspot.com/script/2445004

Map - Retrieving elements

- retrieving elements
  - def map = [first : "Paul", last : "Woods"]
  - def key = "first"
  - def val1 = map.first
  - def val2 = map["first"]
  - def val3 = map[key]
  - def val4 = map."$key"
  - println "val1 = " + val1
  - println "val2 = " + val2
  - println "val3 = " + val3
  - println "val4 = " + val4

- val1 = Paul
- val2 = Paul
- val3 = Paul
- val4 = Paul

http://groovyconsole.appspot.com/script/2485003

Map - Removing Elements

- Removing elements from a map
  - def map = [first : "Paul", last : "Woods"]
  - map.remove('first')
  - println map
    - [last:Woods]

http://groovyconsole.appspot.com/script/2425007

Map - Find

- finding elements
  - def map = [first : "Paul", last : "Woods"]
  - def result1 = map.find { kv -> kv.value == "Woods" }
  - println result1.getClass()
  - println result1
    - class java.util.LinkedHashMap$Entry
    - last=Woods

http://groovyconsole.appspot.com/script/2525001

Map - FindAll

- finding elements
  - def map = [first : "Paul", middle: “Aexander”, last : "Woods"]
  - def result2 = map.findAll { kv -> kv.key != "last" }
  - println result2.getClass()
  - println result2
    - class java.util.LinkedHashMap
    - [first:Paul, middle:Alexander]

http://groovyconsole.appspot.com/script/2465003

Range

- A Range is a data structure that contains the beginning and ending value of a sequence. It can iterate through the sequence, and determine if a value is inside or outside of the sequence
  - def range = (1..5)
  - println range
  - println range.class
    - [1, 2, 3, 4, 5]
    - class groovy.lang.IntRange

http://groovyconsole.appspot.com/script/2445005

Range - Iteration

- You can use the each method to loop through all of the values.
  - def range = 1..5
  - range.each { println it }
    - 1
    - 2
    - 3
    - 4
    - 5

http://groovyconsole.appspot.com/script/2405006

Range - Iteration - Step

- You can use the each method to loop through all of the values.
  - def range = (1..5)
  - range.step(2) { println it }
    - 1
    - 3
    - 5

http://groovyconsole.appspot.com/script/2525002

Range - Contains

- You can use the each method to loop through all of the values.
  - def range = (1..5)
  - println "contains 5 : " + range.contains(5)
  - println "contains 7 : " + range.contains(7)
    - contains 5 : true
    - contains 7 : false

http://groovyconsole.appspot.com/script/2485004

Range - Data Types

- Ranges can also work on other data types, including dates.
  - def range2 = (new Date()-7 .. new Date())
  - range2.each { date -> println date }
    - Wed Feb 26 13:15:43 CST 2014
    - Thu Feb 27 13:15:43 CST 2014
    - Fri Feb 28 13:15:43 CST 2014
    - Sat Mar 01 13:15:43 CST 2014
    - Sun Mar 02 13:15:43 CST 2014
    - Mon Mar 03 13:15:43 CST 2014
    - Tue Mar 04 13:15:43 CST 2014
    - Wed Mar 05 13:15:43 CST 2014

http://groovyconsole.appspot.com/script/2425008

Operation - ?. subscript

- This operator checks if the value is null, and either returns null or calls the method and returns its result.
- This code fails with NPE
  - def list = ['a', 'b', null, 'c', 'd']
  - list.each { item -> println item.toUpperCase() }
    - A
    - B
    - Caught: java.lang.NullPointerException: Cannot invoke method toUpperCase() on null object

http://groovyconsole.appspot.com/script/2465004

Operation - ?. subscript

- This code succeeds
  - def list = ['a', 'b', null, 'c', 'd']
  - list.each { item -> println item?.toUpperCase() }
    - A
    - B
    - null
    - C
    - D

http://groovyconsole.appspot.com/script/2535001

Operation - ?: conditional - Elvis

- if object is false, return another object. else return the object
  - def a = null
  - def b = ""
  - def c = "abc"
  - println "null : " + (a ?: "it is false")
  - println "empty : " + (b ?: "it is false")
  - println "value : " + (c ?: "it is false")
    - null : it is false
    - empty : it is false
    - value : abc

http://groovyconsole.appspot.com/script/2475004

Operation - <=> - spaceship

- calls the .compareTo method
- returns -1 if a < b
- returns +1 if a > b
- returns 0 if a == b
  - println "1 <=> 5 : " + (1 <=> 5)
  - println “5 <=> 1 : " + (5 <=> 1)
  - println "1 <=> 1 : " + (1 <=> 1)
    - 1 <=> 5 : -1
    - 5 <=> 1 : 1
    - 1 <=> 1 : 0

http://groovyconsole.appspot.com/script/2545001

Closures - Introduction

- A closure is a block of code.
- Unlike methods, the closure is assigned to a object.
- A closure can be reassigned to another object.
- The scope of a method (the values it can access) is determined by who owns the closure (by default). The scope rules can be changed at runtime.

Closures - Create and Call

- Define a closure that adds numbers, and call it
  - Closure add = { a, b ->
  -     a+b
  - }
  - println add(1,2)
    - 3

http://groovyconsole.appspot.com/script/2415003

Closure - Zero Parameter Syntax

- Syntax for zero parameter closures
  - def zero = { ->
  -   println "zero parameters"
  - }
  - zero.call()
    - zero parameters

http://groovyconsole.appspot.com/script/2515002

Closure - One Parameter Syntax

- Syntax for a one parameter closure.
  - def one_a = {
  -   println "one parameter : $it"
  - }
  - def one_b = { a->
  -   println "one named parameter : $a"
  - }
  - one_a.call('alpha')
  - one_b.call('beta')
    - one parameter : alpha
    - one named parameter : beta

http://groovyconsole.appspot.com/script/2555001

Closure - Parameter Syntax

- Syntax for 2+ parameter closures
  - def two = { a, b ->
  -   println "two parameters : $a $b"
  - }
  - two.call('22', '2222')
    - two parameters : 22 2222

http://groovyconsole.appspot.com/script/2515003

Closure - Method Takes Closure

- A method that takes a closure
  - class Name {
  -     def first
  -     def modify(Closure closure) {
  -         closure.call this
  -     }
  -     String toString() {first}
  - }
  - def capitalizer = { Name name ->
  -     name.first = name.first.capitalize()
  - }
  - def paul = new Name(first:"paul")
  - println "before = " + paul
  - paul.modify capitalizer
  - println "after = " + paul
    - before = paul
    - after = Paul

http://groovyconsole.appspot.com/script/2525003

Closure - Delegate

- Delegate changes the scope of a closure
  - class Name {
  -     def name = ""
  - }
  - def name1 = new Name(name:"Paul")
  - def name2 = new Name(name:"Woods")

  - def namePrinter = { ->
  -     println name
  - }

  - namePrinter.delegate = name1
  - namePrinter()
  - namePrinter.delegate = name2
  - namePrinter()
    - Paul
    - Woods

http://groovyconsole.appspot.com/script/2405007

MultiAssign

- Initialize or assign multiple variables with values from a list.
  - def a
  - def b
  - (a, b) = [1, 2]
  - def (c, d) = [3 , 4]

  - println "a=$a"
  - println "b=$b"
  - println "c=$c"
  - println "d=$d"
    - a=1
    - b=2
    - c=3
    - d=4

http://groovyconsole.appspot.com/script/2465005

Optional parenthesis, semicolons, and returns

- In some situations, groovy allows you to remove parenthesis, semicolons and return statements.

Optional - Parenthesis 1

- No Arguments and no 'get' prefix - () mandatory
  - class Name {
  -     def first, last
  -     def print() { println first + " " + last }
  -     def printDelim(delim) { println first + delim + last }
  -     def getFullName() { return first + " " + last }
  -     def getTotal(delim) { return first + delim + last }
  - }
  - def name = new Name(first:"Paul", last:"Woods")
  - name.print()   // () required
    - Paul Woods

http://groovyconsole.appspot.com/script/2455003

Optional - Parenthesis 2

- One or more arguments and not referencing the return value - () optional
  - class Name {
  -     def first, last
  -     def print() { println first + " " + last }
  -     def printDelim(delim) { println first + delim + last }
  -     def getFullName() { return first + " " + last }
  -     def getTotal(delim) { return first + delim + last }
  - }
  - def name = new Name(first:"Paul", last:"Woods")

  - name.printDelim " "   // () not required
    - Paul Woods

http://groovyconsole.appspot.com/script/2555002

Optional - Parenthesis 3

- The method has a ‘get’ prefix, and no arguments. () optional
  - class Name {
  -     def first, last
  -     def print() { println first + " " + last }
  -     def printDelim(delim) { println first + delim + last }
  -     def getFullName() { return first + " " + last }
  -     def getTotal(delim) { return first + delim + last }
  - }
  - def name = new Name(first:"Paul", last:"Woods")
  - println name.fullName   // drop get prefix and ()
    - Paul Woods

http://groovyconsole.appspot.com/script/2495005

Optional - Parenthesis 4

- Method has 'get' prefix and 1 or more arguments and using the return value. () mandatory
  - class Name {
  -     def first, last
  -     def print() { println first + " " + last }
  -     def printDelim(delim) { println first + delim + last }
  -     def getFullName() { return first + " " + last }
  -     def getTotal(delim) { return first + delim + last }
  - }
  - def name = new Name(first:"Paul", last:"Woods")
  - println name.getTotal(",")   // () mandatory
    - Paul Woods

http://groovyconsole.appspot.com/script/2545002

Optional - Semicolons

- Semicolons are almost always optional
- Must be used if multiple statements on a single line.
  - def a = 1
  - def b = 2
  - println a
  - println b
  - println a; println b
    - 1
    - 2
    - 1
    - 2

http://groovyconsole.appspot.com/script/2435007

Optional - Returns - 1

- Returns are optional when the value to be returned is the last line of the method.

```
def sum(a, b) {
   a + b
}
def sub(a, b) {
   def total = a - b
   total
}
println "sum 1 and 2 = " + sum(1,2)
println "sub 9 and 3 = " + sub(9,3)
```

http://groovyconsole.appspot.com/script/2495006

Optional - Returns - 2

- Returns are optional when the method is a if/else method. The value to be returned is the last line of each block, and the if/else is the bottom of the method.

```
def choose(a, b, c) {
    if(a > 0) {
        b
    } else if(a < 0) {
        c
    } else {
        0
    }
}
println " 1 : " + choose( 1, 10, 20)
println "-1 : " + choose(-1, 10, 20)
println " 0 : " + choose( 0, 10, 20)
```

```
 1 : 10
-1 : 20
 0 : 0
```

http://groovyconsole.appspot.com/script/2415004

PowerAssert

- Power Assert - in a failed assert statement, groovy shows you the values of the objects.
  - def map = [a: [b: [c: 2]]]
  - assert 3 == map.a.b.c

```
Assertion failed:

assert 3 == map.a.b.c
         |  |   | | |
         |  |   | | 2
         |  |   | [c:2]
         |  |   [b:[c:2]]
         |  [a:[b:[c:2]]]
         false
```

http://groovyconsole.appspot.com/script/2505002

PowerAssert - Gotcha 1

- PowerAssert Gotcha - If the difference is leading/trailing white space or control characters, the assert won't display a difference.
  - def a = "a"
  - def b = "a\r\n"
  - assert a == b

```
Assertion failed:
assert a == b
       | |  |
       a |  a
         false
```

- It fails, but you can't tell why

Meta Programming

- Groovy can dynamically modify the code of classes at runtime

MP - Add Method to Object

- Adding a method to a object – only that object can use it
  - String a = "a"
  - String b = "b"
  - a.metaClass.hashIt = { ->
  -   "#" + delegate + "#"
  - }
  - println a.hashIt()
  - println b.hashIt()
    - #a#
    - Caught: groovy.lang.MissingMethodException: No signature of method: java.lang.String.hashIt() …

http://groovyconsole.appspot.com/script/2485005

MP - Add Method to Class

- Adding a method to a class – all objects of the class can use it.
  - String a = "a"
  - String b = "b"
  - String.metaClass.hashIt = { ->
  -   "#" + delegate + "#"
  - }
  - println a.hashIt()
  - println b.hashIt()
    - #a#
    - #b#

http://groovyconsole.appspot.com/script/2475005

MP - Add Static Method to Class

- Adding a method to a class
  - String a = "a"
  - String.metaClass.static.hashIt = { ->
  -    "#" + delegate + "#"
  - }
  - println a.hashIt()
  - println String.hashIt()
    - #a#
    - #class java.lang.String#

http://groovyconsole.appspot.com/script/2495007

Conclusion

- Read the Groovy JDK to see what Groovy added to the java classes.
  - http://groovy.codehaus.org/groovy-jdk/
- Read about the groovy transformation annotations
  - http://groovy.codehaus.org/gapi/index.html?groovy/transform/ToString.html
- Try Grails - Groovy / Spring / Hibernate WebApp
  - http://grails.org/
- Try Gradle - Groovy Build Automation
  - http://gradle.org/
- Try Gaelyk - Groovy on Google AppServer
  - http://gaelyk.appspot.com/
- Try Griffon - Groovy desktop java applications
  - http://griffon.codehaus.org/
- Try Gpars - Concurrency with Groovy
  - http://gpars.codehaus.org/
- Abstract Syntax Trees
  - http://groovy.codehaus.org/Compile-time+Metaprogramming+-+AST+Transformations